POD: A Parallel-On-Die Architecture

Dong Hyuk Woo
Joshua B. Fryman
Allan D. Knies
Marsha Eng
Hsien-Hsin S. Lee

Georgia Tech
Intel Corp.
Intel Berkeley Research
Intel Corp.
Georgia Tech
What So New with ‘Many-Core’?

• On Chip!
  – New definition of scalability!

Performance Scalability
- Minimize communication
- Hide communication latency

Performance Scalability
- Minimize communication
- Hide communication latency

Area Scalability
- How many cores on a die?

Power Scalability
- How much power per core?
## Scalable Power Consumption?

<table>
<thead>
<tr>
<th>Technology (nm)</th>
<th>65</th>
<th>45</th>
<th>32</th>
<th>22</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td># of cores</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>Frequency (GHz)</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
</tr>
<tr>
<td>Vdd (V)</td>
<td>1.3</td>
<td>1.1</td>
<td>0.9</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>Power / core (W)</td>
<td>37.5</td>
<td>26.8</td>
<td>18.0</td>
<td>14.2</td>
<td>10.9</td>
</tr>
<tr>
<td>Total power (W)</td>
<td>150</td>
<td>215</td>
<td>288</td>
<td>454</td>
<td>696</td>
</tr>
</tbody>
</table>

* The above pictures do not represent any current or future Intel products.
Thousand Cores, Power Consumption?
Some Known Facts on Interconnection

• One main energy hog!
  – Alpha 21364:
    • 20% (25W / 125W)

  – MIT RAW:
    • 40% of individual tile power

  – UT TRIPS:
    • 25%

• Mesh-based MIMD many-core?
  – Unsustainable energy in its router logic
  – Unpredictable communication pattern
    • Unfriendly to low-power techniques
POD: A Parallel-On-Die Architecture
Design Goals

- Broad-purpose acceleration
  - Scalable vector performance
- Energy and area efficiency
- Low latency inter-PE communication
- Best-in-class scalar performance
- Backward compatibility
Host Processor
Processing Elements (PEs)
Instruction Bus (IBus)
ORTree: Status Monitoring

Flag status

Host Processor
Data Return Buffer (DRB)
2D Torus P2P Links
2D Torus P2P Links
RRQ & MBus: DMA

Hsien-Hsin Sean Lee, POD: A Parallel-On-Die Architecture
On-chip MC / Ring

Host Processor

System memory

MC

RRQ

RRQ

RRQ

RRQ

RRQ

Memory Controller

DRB

DRB

DRB

DRB

Arbitrator

ARB

xTLB

LLC
Wire Delay Aware Design
Simplified PE Pipelining

96-bit VLIW instruction

DEC / RF

GBUS

IBUS

MBUS 80

NSEW

Local SRAM

Decoded Instruction
Physical Design Evaluation

![Intel Conroe Core 2 Duo processor die photo](image-url)
Physical Design Evaluation

Intel Conroe Core 2 Duo processor die photo
Physical Design Evaluation @ 45nm

- **PE**
  - 128KB dual-ported SRAM, Basic GP ALU, simplified SSE ALU
  - ~3.2 mm²

- **RRQ**
  - ~3.2 mm² (conservative estimation)

- **The host processor**
  - Core-based microarchitecture
  - ~25.9 mm²

- **3MB LLC**
  - ~20 mm²

- **Total**: ~276.5 mm²
Interconnection Fabric Among PEs

• Different links for different communication
  – IBus / Mbus / 2D torus P2P links / ORTree
  – No contention among different communication

• Fully synchronized and orchestrated by the host processor (and RRQ for MBus)
  – No crossbar
  – No contention / arbitration
  – No buffer space for the router
  – Nothing unpredictable!
Design Space Exploration

- Baseline Design
Design Space Exploration

- **3D POD with many-core**

You live and **work** here

You get your **beer** here
Design Space Exploration

- 3D many-POD processor
Programming Model
Programming Model

- Example code: Reduction

\[ S = \sum_{i=0}^{N-1} a_i \]

```c
char* base_addr = (char*) malloc(npe);
for ( i=0; I < npe; i++ ) {
    *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14
$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
    $ASM xfer.east r1 = r1
    $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
    $ASM xfer.north r1 = r1
    $ASM add r2 = r2, r1
}
```
Performance Result

<table>
<thead>
<tr>
<th>Operation</th>
<th>1x1 POD</th>
<th>2x2 POD</th>
<th>4x4 POD</th>
<th>8x8 POD</th>
</tr>
</thead>
<tbody>
<tr>
<td>DenseMMM (SP)</td>
<td>860.3 GFLOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFT (SP)</td>
<td>91.4 GFLOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDCT (DP)</td>
<td>504.6 GFLOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option Pricing (SP)</td>
<td>134.8 GFLOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Down Sampling (SP)</td>
<td>113.6 GFLOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-means (SP)</td>
<td>869.3 GFLOPS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance per mm$^2$

- **Homo-geneous many-core (ideal)**

The chart shows performance per mm$^2$ for various algorithms (DenseMMM, FFT, IDCT, Option Pricing, Down Sampling, K-means) across different POD configurations (1x1 POD, 2x2 POD, 4x4 POD, 8x8 POD) for 1 core, 4 cores, 16 cores, and 64 cores.
Performance per Watt

Performance per Watt vs. Pricing and Sampling for different POD configurations.

Legend:
- 1x1 POD
- 2x2 POD
- 4x4 POD
- 8x8 POD

DenseMMM, FFT, IDCT, Option Pricing, Down Sampling, K-means, Homogeneous many-core (ideal)
Performance per Joule

DenseMMM  FFT  IDCT  Option Pricing  Down Sampling  K-means  Homogeneous many-core (ideal)

Performance per Joule

<table>
<thead>
<tr>
<th>p</th>
<th>1x1 POD</th>
<th>2x2 POD</th>
<th>4x4 POD</th>
<th>8x8 POD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.125</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.250</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.500</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 core  16 cores  64 cores
## Interconnection Power

<table>
<thead>
<tr>
<th></th>
<th>Input buffer</th>
<th>X-bar</th>
<th>arbiter</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>31%</td>
<td>30%</td>
<td>~0%</td>
<td>39%</td>
</tr>
<tr>
<td>TRIPS</td>
<td>35%</td>
<td>33%</td>
<td>1%</td>
<td>31%</td>
</tr>
</tbody>
</table>

Wang et al., “Power-Driven Design of Router Microarchitectures in On-Chip Networks”, MICRO 2003
2D Torus P2P Link Active Time

Hsien-Hsin Sean Lee, POD: A Parallel-On-Die Architecture
Conclusion

• We propose POD
  – Parallel-On-Die many-core architecture
  – Broad-purpose acceleration
  – High energy and area efficiency
  – Backward compatibility

• A single-chip POD can achieve up to 1.5 TFLOPS of IEEE SP FP operations.

• Interconnection architecture of POD is extremely power- and area-efficient.
Let’s Use Power to Compute, Not Commute!

Georgia Tech
ECE MARS Labs
http://arch.ece.gatech.edu
Back-up Slides
PODSIM

Native x86 machine

PE pipelining
Memory subsystem
Accurate modeling of on-chip, off-chip bandwidth

Limitation:
Host processor overhead not modeled (I$ miss, branch misprediction)
LLC takeover of the MC ring by the host
Why multi-processing again?

- **No more faster processor**
  - Power wall
  - Increasing wire delay
  - Diminishing return of deeply pipelined architecture

- **No more illusion of a sequential processor**
  - Design complexity
  - Diminishing return of wide-issue superscalar processors
Out-of-order Host Issues

• Speculative execution
  – No recovery mechanism in each PE
  – Broadcast POD instructions in a non-speculative manner
    • As long as host-side code does not depend on the results from POD, this approach does not affect the performance.

• Instruction Reordering
  – POD instructions are strongly ordered.
    • IBits queue
      – Similar to the store queue
x86 Modification

• 5 new instructions:
  – sendibits, getort, drainort, movl, getdrb,

• 3 modified instructions:
  – lfence, sfence, mfence

• 4 possible new co-processor registers or else mem-mapped addr:
  – ibits register, ort status, drb value, xTLB support

• IBits queue

• All other modifications external to the x86 core
  – Arbiter for LLC priority can be external to LLC

• Deferred Design Issue: LLC-POD Coherence
  – v1 Study: uncachable
  – v2 Prototype: LLC Snoops on A/D rings
POD ISA

• “Instructions” are 3-wide VLIW bundles,
  – One of each G,X,M (32 bits each)

• Integer Pipe (G):
  – and,or,not,xor,shl,shr,add,sub,mul,cmp,cmov,xfer,xferdrb : !div

• SSE Pipe (X):
  – pfp{fma,add,sub,mul,div,max,min},pi{add,sub,mul,and,or,not,cmp},shuffle,cvt,xfer : !idiv

• Memory Pipe (M):
  – ld,st,blkrd/wr,mask{set,pop,push},mov,xfer

• Hybrid (G+X+M):
  – movl
Power Scalability!

- Graph showing relative performance per joule versus relative chip power budget.
- Three color-coded grid patterns in the background, possibly representing different architectures or stages of a process.

Hsien-Hsin Sean Lee, POD: A Parallel-On-Die Architecture
Some Known Facts on Interconnection

- **MIT RAW**
  - ~ 40% of die area

Some Known Facts on Interconnection

• One main energy hog!
  – **Alpha 21364:**
    • 20% (25W / 125W)
  – **MIT RAW:**
    • 40% of individual tile power
  – **UT TRIPS:**
    • 25%

Wang et al., “Power-Driven Design of Router Microarchitectures in On-Chip Networks”, MICRO 2003
Some Known Facts on Interconnection

- Mesh-based MIMD many-core?
  - Unsustainable energy in its router logic
  - Unpredictable communication pattern
    - Difficult to apply common low-power techniques

Programming Model
Programming Model

• Example code: Reduction

\[ S = \sum_{i=0}^{N-1} a_i \]

```c
char* base_addr = (char*) malloc(npe);
for (i=0; i < npe; i++) {
    *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14
$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for (i=0; i < (npeX-1); i++) {
    $ASM xfer.east r1 = r1
    $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for (i=0; i < (npeY-1); i++) {
    $ASM xfer.north r1 = r1
    $ASM add r2 = r2, r1
}
```
Programming Model (2x2 POD)

- Malloc memory at the host memory space

```c
char* base_addr = (char*) malloc(npe);
for (i=0; i < npe; i++) {
    *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14
$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for (i=0; i < (npeX-1); i++) {
    $ASM xfer.east r1 = r1
    $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for (i=0; i < (npeY-1); i++) {
    $ASM xfer.north r1 = r1
    $ASM add r2 = r2, r1
}
```
Programming Model (2x2 POD)

- Initialize $a_i$

```c
char* base_addr = (char*) malloc(npe);
for ( i=0; i < npe; i++ ) {
    *(base_addr+i) = i+1;
}

$ASM\ mov\ r15 = MY\_PE\_ID\_POINTER
$ASM\ ld\ r15 = local [ r15 + 0 ]
POD\_movl( r14, base\_addr );

$ASM\ add\ r15 = r15, r14

$ASM\ ld\ r1 = sys [ r15 + 0 ]
POD\_mfence();

$ASM\ add\ r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
    $ASM\ xfer\_east\ r1 = r1
    $ASM\ add\ r2 = r2, r1
}

$ASM\ add\ r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
    $ASM\ xfer\_north\ r1 = r1
    $ASM\ add\ r2 = r2, r1
}
```
Programming Model (2x2 POD)

- Load the address of my PE ID

```c
char* base_addr = (char*) malloc(npe);
for (i=0; i < npe; i++) {
    *(base_addr+i) = i+1;
}

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local[r15 + 0]
POD_movl(r14, base_addr);

$ASM add r15 = r15, r14

$ASM ld r1 = sys[r15 + 0]
POD_mfence();

$ASM add r2 = r2, r1
for (i=0; i < (npeX-1); i++) {
    $ASM xfer.east r1 = r1
    $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for (i=0; i < (npeY-1); i++) {
    $ASM xfer.north r1 = r1
    $ASM add r2 = r2, r1
}
```
Programming Model (2x2 POD)

• Load my PE ID

char* base_addr = (char*) malloc(npe);
for ( i=0; i < npe; i++ ) {
    *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14
$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i < (npeX-1); i++ ) {
    $ASM xfer.east r1 = r1
    $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i < (npeY-1); i++ ) {
    $ASM xfer.north r1 = r1
    $ASM add r2 = r2, r1
}
Programming Model (2x2 POD)

• Broadcast base_addr

```c
char* base_addr = (char*) malloc(npe);
for ( i=0; i < npe; i++ ) {
    *(base_addr+i) = i+1;
}

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14
$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
    $ASM xfer.east r1 = r1
    $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
    $ASM xfer.north r1 = r1
    $ASM add r2 = r2, r1
}
```
Programming Model (2x2 POD)

- Calculate the pointer to my $a_i$

```c
char* base_addr = (char*) malloc(npe);
for ( i=0; i < npe; i++ ) {
    *(base_addr+i) = i+1;
}

$ASM$ mov r15 = MY_PE_ID_POINTER
$ASM$ ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM$ add r15 = r15, r14

$ASM$ ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM$ add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
    $ASM$ xfer.east r1 = r1
    $ASM$ add r2 = r2, r1
}
$ASM$ add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
    $ASM$ xfer.north r1 = r1
    $ASM$ add r2 = r2, r1
}
```
Programming Model (2x2 POD)

- Load my $a_i$ from the host memory space

```c
char* base_addr = (char*) malloc(npe);
for ( i=0; i < npe; i++ ) {
    *(base_addr+i) = i+1;
}

$ASM\ mov r15 = MY\_PE\_ID\_POINTER$
$ASM\ ld r15 = local [ r15 + 0 ]$
POD_movl( r14, base_addr );

$ASM\ add r15 = r15, r14$

$ASM\ ld r1 = sys [ r15 + 0 ]$
POD_movf();

$ASM\ add r2 = r2, r1$
for ( i=0; i < (npeX-1); i++ ) {
    $ASM\ xfer.east r1 = r1$
    $ASM\ add r2 = r2, r1$
}

$ASM\ add r1 = r2, 0$
for ( i=0; i < (npeY-1); i++ ) {
    $ASM\ xfer.north r1 = r1$
    $ASM\ add r2 = r2, r1$
}
```
Programming Model (2x2 POD)

- Wait until it is loaded

```c
char* base_addr = (char*) malloc(npe);
for ( i=0; i < npe; i++ ) {
  *(base_addr+i) = i+1;
}

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14
$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
  $ASM xfer.east r1 = r1
  $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
  $ASM xfer.north r1 = r1
  $ASM add r2 = r2, r1
}
```

Hsien-Hsin Sean Lee, POD: A Parallel-On-Die Architecture
Programming Model (2x2 POD)

- Update S

```c
char* base_addr = (char*) malloc(npe);
for ( i=0; i < npe; i++ ) {
    *(base_addr+i) = i+1;
}

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i < (npeX-1); i++ ) {
    $ASM xfer.east r1 = r1
    $ASM add r2 = r2, r1
}

$ASM add r1 = r2, 0
for ( i=0; i < (npeY-1); i++ ) {
    $ASM xfer.north r1 = r1
    $ASM add r2 = r2, r1
}
```
Programming Model (2x2 POD)

• 1st iteration

```c
char* base_addr = (char*) malloc(npe);
for ( i=0; i < npe; i++ ) {
    *(base_addr+i) = i+1;
}

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i < (npe-1); i++ ) {
    $ASM xfer.east r1 = r1
    $ASM add r2 = r2, r1
}

$ASM add r1 = r2, 0
for ( i=0; i < (npeY-1); i++ ) {
    $ASM xfer.north r1 = r1
    $ASM add r2 = r2, r1
}
```

Hsien-Hsin Sean Lee, POD: A Parallel-On-Die Architecture
Programming Model (2x2 POD)

- Transfer my $a_i$ to my **east-side** neighbor

```c
char* base_addr = (char*) malloc(npe);
for ( i=0; i < npe; i++ ) {
    *(base_addr+i) = i+1;
}
$ASM \text{ mov } r15 = \text{MY\_PE\_ID\_POINTER} \\
$ASM \text{ ld } r15 = \text{local } [ r15 + 0 ] \\
POD\_movl( r14, base_addr );
$ASM \text{ add } r15 = r15, r14 \\
$ASM \text{ ld } r1 = \text{sys } [ r15 + 0 ] \\
POD\_mfence();
$ASM \text{ add } r2 = r2, r1 \\
for ( i=0; i < (npe\_X-1); i++ ) { \\
    $ASM \text{ xfer.east } r1 = r1 \\
    $ASM \text{ add } r2 = r2, r1 \\
} \\
$ASM \text{ add } r1 = r2, 0 \\
for ( i=0; i < (npe\_Y-1); i++ ) { \\
    $ASM \text{ xfer.north } r1 = r1 \\
    $ASM \text{ add } r2 = r2, r1 \\
} \\
```
Programming Model (2x2 POD)

- Update S

```c
char* base_addr = (char*) malloc(npe);
for ( i=0; i < npe; i++ ) {
    *(base_addr+i) = i+1;
}

ASM mov r15 = MY_PE_ID_POINTER
ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

ASM add r15 = r15, r14

ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
    $ASM xfer.east r1 = r1
    $ASM add r2 = r2, r1
}

ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
    $ASM xfer.north r1 = r1
    $ASM add r2 = r2, r1
}
```
Programming Model (2x2 POD)

• Copy rowsum to r1

```
char* base_addr = (char*) malloc(npe);
for ( i=0; i < npe; i++ ) {
    *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );
$ASM add r15 = r15, r14
$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();
$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
    $ASM xfer.east r1 = r1
    $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
    $ASM xfer.north r1 = r1
    $ASM add r2 = r2, r1
}
```

Hsien-Hsin Sean Lee, POD: A Parallel-On-Die Architecture
Programming Model (2x2 POD)

• 1st iteration

char* base_addr = (char*) malloc(npe);
for ( i=0; i < npe; i++ ) {
    *(base_addr+i) = i+1;
}

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i < (npeX-1); i++ ) {
    $ASM xfer.east r1 = r1
    $ASM add r2 = r2, r1
}

$ASM add r1 = r2, 0
for ( i=0; i < (npeY-1); i++ ) {
    $ASM xfer.north r1 = r1
    $ASM add r2 = r2, r1
}
Programming Model (2x2 POD)

- **Transfer my rowsum to my north-side neighbor**

```c
char* base_addr = (char*) malloc(npe);
for ( i=0; i < npe; i++ ) {
    *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );
$ASM add r15 = r15, r14
$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();
$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
    $ASM xfer.east r1 = r1
    $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
    $ASM xfer.north r1 = r1
    $ASM add r2 = r2, r1
}
```

---

Hsien-Hsin Sean Lee, POD: A Parallel-On-Die Architecture
Programming Model (2x2 POD)

• Update S

```c
char* base_addr = (char*) malloc(npe);
for ( i=0; i < npe; i++ ) {
    *(base_addr+i) = i+1;
}

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );

$ASM add r15 = r15, r14

$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();

$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
    $ASM xfer.east r1 = r1
    $ASM add r2 = r2, r1
}

$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
    $ASM xfer.north r1 = r1
    $ASM add r2 = r2, r1
}
```

Hsien-Hsin Sean Lee, POD: A Parallel-On-Die Architecture
Programming Model (2x2 POD)

• Done!

```
char* base_addr = (char*) malloc(npe);
for ( i=0; i < npe; i++ ) {
    *(base_addr+i) = i+1;
}
$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]
POD_movl( r14, base_addr );
$ASM add r15 = r15, r14
$ASM ld r1 = sys [ r15 + 0 ]
POD_mfence();
$ASM add r2 = r2, r1
for ( i=0; i< (npeX-1); i++ ) {
    $ASM xfer.east r1 = r1
    $ASM add r2 = r2, r1
}
$ASM add r1 = r2, 0
for ( i=0; i< (npeY-1); i++ ) {
    $ASM xfer.north r1 = r1
    $ASM add r2 = r2, r1
}
```

Hsien-Hsin Sean Lee, POD: A Parallel-On-Die Architecture
Programming Model (2x2 POD)

- Broadcast the pointer to ‘result’

  int result;

  POD_movl( r14, &result);

  $ASM \text{ mov } r15 = \text{MY\_PE\_ID\_POINTER}$
  $ASM \text{ ld } r15 = \text{local } [ r15 + 0 ]$

  $ASM \text{ sub } r15 = r15, 0$
  $ASM \text{ pushmask.e}$

  $ASM \text{ st } \text{sys}[ r14 + 0 ] = r15$
  $ASM \text{ popmask}$

  POD_mfence();

  base_addr: 16

  16: 1 128: <- addr of ‘result’
  17: 2
  18: 3
  19: 4
Programming Model (2x2 POD)

• Only PE0 will write-back!

```c
int result;

POD_movl( r14, &result );

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]

$ASM sub r15 = r15, 0
$ASM pushmask.e
    $ASM st sys[ r14 + 0 ] = r15
$ASM popmask

POD_mfence();
```

base_addr: 16

16: 1  128: <- addr of ‘result’
17: 2
18: 3
19: 4
Programming Model (2x2 POD)

• Load PE_ID

```c
int result;
POD_movl( r14, &result );

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]

$ASM sub r15 = r15, 0
$ASM pushmask.e
  $ASM st sys[ r14 + 0 ] = r15
$ASM popmask

POD_mfence();
```

base_addr: 16

16: 1 128: <- addr of ‘result’
17: 2
18: 3
19: 4
Programming Model (2x2 POD)

- Compare PE_ID

```c
int result;

POD_movl(r14, &result);

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]

$ASM sub r15 = r15, 0
$ASM pushmask.e
  $ASM st sys[ r14 + 0 ] = r15
$ASM popmask

POD_mfence();
```

base_addr: 16

16: 1 128: <- addr of 'result'
17: 2
18: 3
19: 4
Programming Model (2x2 POD)

• Enable PE only when equal to zero

```c
int result;
POD_movl( r14, &result );

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]

$ASM sub r15 = r15, 0
$ASM pushmask.e
    $ASM st sys[ r14 + 0 ] = r15
$ASM popmask

POD_mfence();
```

base_addr: 16

16: 1  128: <- addr of ‘result’
17: 2
18: 3
19: 4
Programming Model (2x2 POD)

- Write-back

```c
int result;

POD_movl( r14, &result );

$ASM mov r15 = MY_PE_ID_POINTER
$ASM ld r15 = local [ r15 + 0 ]

$ASM sub r15 = r15, 0
$ASM pushmask.e
$ASM st sys[ r14 + 0 ] = r15
$ASM popmask

POD_mfence();
```

```
base_addr: 16

16: 1 128: <- addr of ‘result’
17: 2
18: 3
19: 4
```
Programming Model (2x2 POD)

- **Enable ALL PEs**

```plaintext
int result;

POD_movl( r14, &result );

$ASM \text{mov } r15 = \text{MY\_PE\_ID\_POINTER}
$ASM \text{ld } r15 = \text{local [ r15 + 0 ]}

$ASM \text{sub } r15 = r15, 0
$ASM \text{pushmask.e}
    $ASM \text{st sys [ r14 + 0 ]} = r15

$ASM \text{popmask}

POD_mfence();
```

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>3</td>
<td>r1</td>
<td>3</td>
</tr>
<tr>
<td>r2</td>
<td>10</td>
<td>r2</td>
<td>10</td>
</tr>
<tr>
<td>r14</td>
<td>128</td>
<td>r14</td>
<td>128</td>
</tr>
<tr>
<td>r15</td>
<td>0</td>
<td>r15</td>
<td>3</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>7</td>
<td>r1</td>
<td>7</td>
</tr>
<tr>
<td>r2</td>
<td>10</td>
<td>r2</td>
<td>10</td>
</tr>
<tr>
<td>r14</td>
<td>128</td>
<td>r14</td>
<td>128</td>
</tr>
<tr>
<td>r15</td>
<td>1</td>
<td>r15</td>
<td>1</td>
</tr>
</tbody>
</table>

base_addr: 16

16: 1 128: <- addr of ‘result’

17: 2
18: 3
19: 4
**Programming Model (2x2 POD)**

- **Wait until the result is written-back**

```c
int result;

POD_movl( r14, &result );

ASM mov r15 = MY_PE_ID POINTER
ASM ld r15 = local [ r15 + 0 ]

ASM sub r15 = r15, 0
ASM pushmask.e
    ASM st sys[ r14 + 0 ] = r15
ASM popmask

POD_mfence();
```

**base_addr: 16**

16: 1  128: <- addr of ‘result’
17: 2
18: 3
19: 4